

PLP - Primer Parcial - 1^{er} cuatrimestre de 2022

Este examen se aprueba obteniendo al menos dos ejercicios bien menos (B-) y uno regular (R). Las notas para cada ejercicio son: -, I, R, B-, B. Entregar cada ejercicio en hojas separadas. Poner nombre, apellido, número de orden y cantidad de hojas en la primera hoja, y numerar las hojas. Se puede utilizar todo lo definido en las prácticas y todo lo que se dio en clase, colocando referencias claras.

El orden de los ejercicios es arbitrario. Recomendamos leer el parcial completo antes de empezar a resolver.

Ejercicio 1 - Programación funcional

En este ejercicio **no** se permite el uso de recursión explícita, a menos que se indique lo contrario.

Considerar la siguiente definición en Haskell que implementa las proposiciones lógicas:

```
data Prop = Var String | Not Prop | And Prop Prop | Or Prop Prop
```

donde:

- **Var x** representa a una variable proposicional con nombre **x**.
- **Not q** representa la negación de una proposición **q**.
- **And p q** representa la conjunción de dos proposiciones **p** y **q**.
- **Or p q** representa la disyunción de dos proposiciones **p** y **q**.

Modelaremos una valuación como una función que asigna a cada una de sus variables un valor de verdad:

```
type Valuacion = String -> Bool
```

Se pide:

a) Definir y dar el tipo de las siguientes funciones. Por tratarse de esquemas de recursión, para definir estas funciones se permite utilizar **recursión explícita**.

- 1) `recProp :: (String -> b) -> (Prop -> b -> b) -> (Prop -> Prop -> b -> b -> b) -> (Prop -> Prop -> b -> b -> b) -> Prop -> b`, que representa el esquema de recursión primitiva `recr` sobre `Prop`.
- ii) `foldProp`, que representa el esquema de recursión estructural `foldr` sobre `Prop` (dar su tipo). Notar que puede definirse en términos de `recProp`.

b) Definir las siguientes funciones indicando qué esquema recursivo utiliza y por qué.

- i) Definir la función `evaluar :: Valuacion -> Prop -> Bool`, que dadas una valuación y una proposición devuelve el valor de verdad de la proposición para esa valuación.
- ii) `negProp :: Prop -> Prop` que dada una proposición devuelve una proposición equivalente a su negación, con la restricción de que no se debe introducir el operador `Not` a menos que lo que se esté negando sea una variable proposicional. Para esto, se deben usar las leyes de De Morgan. Por ejemplo:

```
> negProp (Not $ Var "q")           > negProp (And (Var "p") (Not (Var "q")))
Var "q"                             Or (Not (Var "p")) (Var "q")
```

Ejercicio 2 - Cálculo lambda

Se desea extender el cálculo lambda con el tipo $AHD_{\sigma, \tau}$, que representa un árbol binario no vacío cuyos nodos internos pueden tener datos de un tipo diferente al de sus hojas. (AHD = árbol con hojas distinguidas).

$\sigma ::= \dots \mid AHD_{\sigma, \sigma}$

$M ::= hoja_{\sigma, \tau}(M) \mid rama(M, M) \mid bin(M, M, M) \mid fold\ M\ hoja_{\sigma} \rightsquigarrow M; rama_{\tau, rec} \rightsquigarrow M; bin_{rec1, r, read} \rightsquigarrow M$

donde:

- $AHD_{\sigma,\tau}$ es el tipo de los árboles con nodos internos de tipo σ y hojas de tipo τ .
- $hoja_{\sigma,\tau}(M)$ representa al $AHD_{\sigma,\tau}$ cuyo único elemento es M (es una hoja).
- $rama(M,N)$ describe al árbol con un nodo interno M y un subárbol N .
- $bin(M,N,O)$ representa al árbol con raíz N y subárboles M y O .
- $fold\ M\ hoja_x \rightsquigarrow N; rama_{n,rec} \rightsquigarrow O; bin_{reci,r,recd} \rightsquigarrow P$ es el esquema de recursión estructural para árboles con hojas distinguidas, donde las variables $x, n, rec, reci, r$ y $recd$ se ligarán a los valores correspondientes según la estructura del árbol M .

Se pide:

- Dar las reglas de tipado para soportar los nuevos términos.
- Describir el nuevo conjunto de valores y dar las reglas de reducción en un paso para los nuevos términos. No es necesario escribir las reglas de congruencia (contextuales), basta con indicar cuántas son.
- Demostrar el siguiente juicio de tipado:

$$\emptyset \triangleright fold\ rama(0, hoja_{Nat, Bool}(True))\ hoja_x \rightsquigarrow True; rama_{n,rec} \rightsquigarrow False; \\ bin_{reci,r,recd} \rightsquigarrow if\ reci\ then\ recd\ else\ False : Bool$$
- Definir como macro la función $mapAHD_{\sigma_1, \tau_1 \rightarrow \sigma_2, \tau_2}$ que, dadas dos funciones de tipos $\sigma_1 \rightarrow \sigma_2$ y $\tau_1 \rightarrow \tau_2$, mapea un AHD_{σ_1, τ_1} a un AHD_{σ_2, τ_2} .

Ejercicio 3 - Inferencia de Tipos

Se desea diseñar un algoritmo de inferencia de tipos para el cálculo lambda extendido con **matrices infinitas** de la siguiente manera:

$\sigma ::= \dots \mid MatInf_{\sigma} \quad M ::= \dots \mid Matriz(M) \mid M[M][M] \leftarrow M \mid M[M][M] \mid map(f, c, x \rightsquigarrow M)(M)$

Estas matrices pueden observarse accediendo a cualquier posición mediante dos números naturales (fila y columna). Pueden también “modificarse” de manera similar, agregando un elemento en una posición, y obteniendo una matriz igual a la original excepto por el elemento contenido en dicha posición. Las matrices se inicializan con un término definido por defecto, el cual será el elemento contenido en las posiciones donde no se haya agregado nada.

Además contamos con una operación especial $map(f, c, x \rightsquigarrow N)(M)$, la cual permite aplicar a cada elemento de la matriz una transformación que depende tanto del elemento en sí - que se liga a la variable x - como de su posición en la matriz - cuya fila y columna se ligán a las variables f y c respectivamente.

Las reglas de tipado son las siguientes:

$$\frac{\Gamma \triangleright M : \sigma}{\Gamma \triangleright Matriz(M) : MatInf_{\sigma}} T-MAT \quad \frac{\Gamma \triangleright M : MatInf_{\sigma} \quad \Gamma \triangleright N : Nat \quad \Gamma \triangleright O : Nat \quad \Gamma \triangleright P : \sigma}{\Gamma \triangleright M[N][O] \leftarrow P : MatInf_{\sigma}} T-INS$$

$$\frac{\Gamma \triangleright M : MatInf_{\sigma} \quad \Gamma \triangleright N : Nat \quad \Gamma \triangleright O : Nat}{\Gamma \triangleright M[N][O] : \sigma} T-OBS$$

$$\frac{\Gamma \triangleright M : MatInf_{\sigma} \quad \Gamma, f : Nat, c : Nat, x : \sigma \triangleright N : \tau}{\Gamma \triangleright map(f, c, x \rightsquigarrow N)(M) : MatInf_{\tau}} T-MAP$$

- Extender el algoritmo de inferencia para admitir las expresiones incorporadas al lenguaje, de tal manera que implemente las reglas de tipado T-MAT, T-INS y T-MAP (no es necesario escribir la extensión del algoritmo para T-OBS).
- Aplicar el algoritmo extendido con el método del árbol para dar el tipo de la siguiente expresión, exhibiendo las sustituciones utilizadas. De no tipar, indicar el motivo.
 $map(f, c, x \rightsquigarrow x\ f)(x[0][Succ(0)] \leftarrow f)$